Examination Computational Statistics

Linköpings Universitet, IDA, Statistik

Course code and name:	732A90 Computational Statistics
Date:	2020/03/24, 8–13
Assisting teacher:	Krzysztof Bartoszek
	All rules are provided in the LISAM instructions for the submission
Grades:	A= $[18-20]$ points
	B= $[16-18)$ points
	C= $[14-16)$ points
	D= $[12-14)$ points
	E= $[10-12)$ points
	F= $[0-10)$ points
Instructions:	Provide a detailed report that includes plots, conclusions and interpretations. If you are unable to include a plot in your solution file clearly indicate the section of R code that generates it. Give motivated answers to the questions. If an answer is not motivated, the points are reduced. Provide all necessary codes in an appendix. In a number of questions you are asked to do plots. Make sure that they are informative, have correctly labelled axes, informative axes limits and are correctly described. Points may be deducted for poorly done graphs. If you have problems with creating a pdf you may submit your solutions in text files with unambiguous references to graphics and code that are saved in separate files There are **TWO** assignments (with sub–questions) to solve. Provide a separate solution file for each assignment. Include all R code that was used to obtain your answers in your solution files. Make sure it is clear which code section corresponds to which question.

NOTE: If you fail to do a part on which subsequent question(s) depend on describe (maybe using dummy data, partial code e.t.c.) how you would do them given you had done that part. You *might* be eligible for partial points.

Assignment 1 (10p)

A random variable has a so called *one–sided strictly stable distribution of order* 1/2 if for a parameter $c > 0$ it has density equalling

$$f(x) = c\sqrt{2\pi}e^{-c^2/(2x)}x^{-3/2}\mathbf{1}_{(0,\infty)}(x).$$

In particular the support is on $(0, \infty)$. The aim of the assignment is to sample from this distribution by using an acceptance–rejection algorithm with the help of a power–law distribution with density

$$f_p(x) = \frac{\alpha - 1}{T_{\min}}\left(\frac{x}{T_{\min}}\right)^{-\alpha}\mathbf{1}_{(T_{\min},\infty)}(x)$$

for $T_{\min} > 0$, $\alpha > 1$. In particular the support is on $(T_{\min}, \infty)$.
TIP: You can obtain the value if π in `R` using `pi`.
TIP: Plot $f(x)$ and think how you can use this.

Question 1.1 (3p)

Can the power–law distribution be used just by itself or is there a problem at any place of the support (**TIP:** there is)? Explain what the problem is and how can it be taken care of.
TIP: recall mixtures—how one can decide from which region of the support to sample from.
TIP: you do not have to solve the issue analytically. You of course may if you want, but this could take a lot of time. It will suffice if you do this numerically and justify your solution (especially the value of a certain number) e.g. graphically.
Provide values of the power–law distribution's parameters that can be used in the acceptance–rejection algorithm. Derive and implement a majorizing density.

Question 1.2 (5p)

Implement an acceptance–rejection algorithm for sampling from the one–sided strictly stable distribution of order 1/2 with the proposal distribution built around the power–law distribution. To sample from a power–law distribution you can use the function `poweRlaw::rplcon()`. If you did not manage to solve the problem indicated in Q.1.1, then implement an "approximate" acceptance–rejection algorithm using a power–law proposal density. In this case explain why and how your output sample deviates from the desired distribution. You may use `runif()` without any restrictions.

Question 1.3 (2p)

Generate a sample of size 50 (you can try more if running time permits) using your implemented sampler, for some choice of c. Present the sample graphically. What is the mean and variance. Now generate samples for other values of c and explore (e.g. graphically) how the mean and variance depend on c.

Assignment 2 (10p)

Your task is to minimize the function

$$f(x_1, x_2) = x_1^2 + \frac{1}{2}x_2^2 + 3$$

using a genetic algorithm.
TIP: As a first step you should always try to visualize the function to minimize. This is a function of two parameters so plotting it might not be straightforward (and is not part of the exam).

Question 2.1 (1p)

Find the minimum of the function analytically.

Question 2.2 (2p)

An initial step when implementing a genetic algorithm is the decision on how individuals in the population will be represented. Often one wants to represent them as binary strings as it makes the implementation of crossover, mutation e.t.c. operators easy.

Each individual is to be represented as a binary vector of length $m = 32$. You may use the following `R` code to transform a binary vector to the integer it represents

```
u<-Reduce(function(s,r) {s*2+r}, v)
```

for example

```
> v<-c(1,0,1)
> Reduce(function(s,r) {s*2+r}, v)
[1] 5
> v<-c(1,0,1,0)
> Reduce(function(s,r) {s*2+r}, v)
[1] 10
```

Notice that `v[1]` is the highest bit.

Each individual is to be represented in the logarithmic coding system

$$x = (-1)^{v[1]} e^{y \cdot (-1)^{v[2]}},$$

where y is the integer represented by `v[3:32]`.

If you are unable to implement this coding of the state space you may use another one of your own choice in order to do the next questions. You *might* be eligible for partial points.

Question 2.2 (3p)

The next step is to implement a selection procedure in order to choose the pool of individuals that will be allowed to contribute to the next generation. Implement a solution that takes advantage of the representation of the numbers and also of the suspected location of the minimum.

Question 2.3 (4p)

Implement all the other necessary components of the genetic algorithm and using it find the minimum of the function $f(x_1, x_2)$. Take a population size of 50. You may not take a constant starting population at the minimum. The starting population has to be random but it may use information on the **approximate (do not use the exact one)** location of the minimum.

It is recommended that you first try your code on a smaller population (say 6) and only after it runs without error, run it on the population of size 50. Follow your population for 100 generations. Provide example calls to your code.

If you did not use at all the information on the approximate location of the minimum, then you risk having very bad populations and long running times.

Visualize the population's behaviour with the number of generations for mutation probability 0.0077 and 0.5 and provide comments. At which generation was the best value found? Was the minimum found? Can you explain the observed behaviour, especially when taking into account the mutation probability?